INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Using Global Behavior Modeling to Improve QoS in Large-scale Distributed Data Storage Services

Jesús Montes , Bogdan Nicolae, Gabriel Antoniu, Alberto Sánchez , María S. Pérez

N° 7271

February 2010

ISSN 0249-6399 ISRN INRIA/RR--7271--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Using Global Behavior Modeling to Improve QoS in Large-scale Distributed Data Storage Services

Jesús Montes *, Bogdan Nicolae†, Gabriel Antoniu‡, Alberto Sánchez §, María S. Pérez*

Thème : Calcul distribué et applications à très haute performance
Équipe KerData

Rapport de recherche n° 7271 — February 2010 — 22 pages

Abstract: As distributed, global-scale, data-intensive applications are becoming more and more common, an increasing pressure is being put on the underlying distributed data services. As such services need to support massively concurrent, largely distributed accesses to huge shared datasets, the stability and scalability of their performance are critical.

More specifically, the ability to sustain a stable high throughput is a very desirable property, as it strongly impacts the quality of service of the data storage system and thereby the overall application performance. Handling quality of service in a large-scale distributed system is however a very difficult task, as a very large number of factors are involved: the data access patterns, the status of a huge number of physical components, etc.

In this paper we explore an approach to the management of quality of service in distributed storage systems based on global behavior modeling combined with client-side quality of service feedback. Our objective is to automate the process of identifying dangerous behavior patterns in storage services.

To demonstrate our approach, we apply GloBeM, a global behavior modeling technique based on monitoring data analysis and machine learning, to improve the quality of service in BlobSeer, a distributed storage for large-scale data-intensive applications specifically designed to sustain high throughput under heavy access concurrency. We evaluate this improvement through extensive evaluations on the Grid'5000 testbed using hard experimental conditions: highly-concurrent data access patterns, for long periods of service uptime, while supporting failures of the physical storage components. Our results show substantial progress in sustaining a higher and more stable data access throughput.

Key-words: Data management; massive data; high throughput; quality of service; machine learning; adaptation to access pattern; BlobSeer; GloBeM

* U. Politécnica de Madrid, Spain.
† University of Rennes 1, France.
‡ INRIA Rennes-Bretagne Atlantique, France.
§ Universidad Rey Juan Carlos, Spain.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Utiliser la modélisation du comportement global pour améliorer la qualité de service dans les systèmes distribués de stockage de données à grande échelle

Résumé : Les applications distribuées à très grande échelle utilisant intensivement les données deviennent de plus en plus nombreuses. Une pression de plus en plus forte est donc exercée sur les services sous-jacents de gestion des données. Ces services doivent permettre des accès massivement concurrent et fortement distribué à de gigantesques ensemble de données. La stabilité et l'extensibilité de leur performance est donc critique.

Plus spécifiquement, leur capacité à soutenir un débit important de manière stable est une propriété très importante. Elle a en effet une forte influence sur la qualité du service rendu par le système de stockage de données et donc sur la performance globale de l'application. Gérer la qualité de service dans un système distribué à grande échelle est cependant une tâche très difficile car un grand nombre de facteurs sont en jeu: les schémas d'accès aux données, l'état d'un très grand nombre de composants, etc.

Dans ce rapport, nous explorons une approche de la gestion de la qualité de service dans des systèmes distribués de stockage fondée sur la modélisation du comportement global combinée avec l'évaluation par le client de la qualité de service. Notre objectif est d'automatiser le processus d'identification des modèles de comportements dangereux dans les services de stockage.

Pour illustrer notre approche, nous présentons GloBeM, une technique de modélisation du comportement global fondée sur l'analyse des données d'observation et l'apprentissage automatique. Cette technique est appliquée au système BlobSeer. BlobSeer est un système de stockage distribué pour les applications de traitement intensif des données à grande échelle, conçu pour permettre un haut débit même en présence d'accès massivement concurrents. Nous évaluons l'amélioration obtenue grâce à une évaluation approfondie sur la plate-forme de test Grid'5000. Cette évaluation se place dans des conditions expérimentales exigeantes: schémas d'accès aux données hautement concurrents pendant de longues durées de service actif, en présence de défaillances au niveau des composants physiques de stockage. Nos résultats confirment une amélioration significative dans le maintien d'un débit important et stable dans l'accès aux données.

Mots-clés : Gestion des données; masses de données; haut débit; qualité de service; apprentissage machine; adaptation aux schémas d'accès; BlobSeer; GloBeM

Contents

1 Introduction

With the emergence of large-scale Internet services, processing massive amounts of data from a broad range of sources (such as web pages, online transaction records, access logs, etc.) becomes more and more common. Such large-scale distributed applications are data-intensive: in a typical scenario, they continuously acquire massive datasets (e.g. based on a streaming pattern) while performing large-scale computations over these (changing) datasets. In order to achieve scalable data processing performance, several paradigms have been proposed, such as MapReduce [6], Dryad [13] and parallel databases [7]. To optimally exploit the data parallelism patterns that are application-specific, such approaches typically force the developer to *explicitly* handle concurrent data accesses. Consequently, these highly-scalable approaches place a heavy burden on the data storage service, which must deal with massively parallel data accesses in an efficient way. Thus, the storage service becomes a critical component on which the whole system performance and scalability depend [9, 12].

To efficiently deal with massively parallel data accesses, the distributed storage service needs to properly address a set of issues such as: *scalable aggregation of storage space* from the participating nodes with minimal overhead, ability to store *huge data objects*, *efficient fine-grain access* to data subsets, *high throughput in spite of heavy access concurrency*, as well as *fault-tolerance*. BlobSeer [18, 19] is an example of distributed storage service for data-intensive applications specifically designed to deal with these issues.

In this paper we focus on an important desirable property that in our view has often been neglected in previous work dedicated to large-scale distributed storage: the *stability of the data throughput*. By properly addressing this issue, a distributed storage service can improves the quality of service for data access. Thereby, the performance of data access becomes much more predictable, which

substantially impacts the overall efficiency of computation scheduling for data-intensive, massively parallel applications.

We must however admit that reasoning about the behavior of the storage service over long periods of time is extremely difficult, especially in a large-scale setting where data-intensive applications are running on infrastructures that are subject to failures (which is a typical situation at large scales). Tailoring the storage service to fit specific application patterns is a non-trivial problem and, even with extensive knowledge of the system, optimizing the storage service becomes virtually impossible. The main cause for this is no other than the sheer complexity of the large-scale system behavior itself.

In this paper we demonstrate how it is possible to automate the process of optimizing a distributed storage service thanks to an innovative approach: *global behavior modeling*. This approach has been introduced through the GloBeM[1] methodology [16, 17], presented in Section 3. We show that this methodology unlocks the potential for significant improvements of the stability of the data access performance. To validate this claim, we perform extensive experimentations in hard conditions: *highly-concurrent data access patterns, for long periods of service uptime, while supporting failures of the physical storage components.*

We summarize our work as follows:

- We apply global behavior modeling to extract valuable, non-trivial information about the behavior observed in the hard conditions above mentioned. The information thus extracted provides a powerful means to synthesize easy-to-use global state information. This is a significant step, given the high complexity inherent to the system itself and to the massive amounts of underlying monitored data. The global state information provides hints on potential bottlenecks in our system with respect to overall throughput and bandwidth stability. Subsequently, we identify optimizations to the BlobSeer's resource allocation policy that address these issues.

- We improve BlobSeer's behavior based on the identified optimizations. We validate the improvement by running large scale experimentations under the same highly concurrent data access pattern and failure scenarios as in the original experiments. We show slight improvements in overall data throughput and substantial improvements in bandwidth stability and in the overall quality of service.

The rest of the paper is organized as follows: in Section 2 we position our approach with respect to related work. We then provide an overview of the GloBeM global behavior modeling approach (Section 3) and of the BlobSeer distributed data service (Section 4). We then describe in detail how the GloBeM approach can be applied to BlobSeer: we present the overall process and the access patterns and failure scenarios used for experiments (Section 5). We report our findings in Section 6 and conclude with a discussion on the significance of our work and on its future extensions.

[1] GloBeM stands for ***Glo**bal **Be**havior **M**odeling*

2 Related work

Modeling and characterizing the behavior of large scale distributed systems has been approached in several other contexts. The most basic approach is benchmarking [8], which enables manual analysis of the behavior of a system under different workloads. Other approaches describe the system formally using Colored Petri Nets (CPN) [2] or Abstract State Machines (ASM) [10] in order to reason about behavior.

Rood and Lewis [22, 23, 24] propose a multi-state model and several analysis techniques in order to forecast the resources availability, aiming at improving scheduler efficiency. The goal of this work is similar to ours, but the main difference lies in the fact that in this case analysis and prediction are performed at resource level. GloBeM analyzes the system focusing on global behavior and single entity aspects.

Li et al. [15] present an Instance Based Learning technique to forecast response times of jobs in large scale systems by means of historical performance data mining. In a similar way, Smith et al. [28] analyze the run times of parallel applications from past executions of similar applications. Cho et al. [4] describe a user demand approach, which employs historical user demands in order to efficiently manage system resources. All these contributions are focused on user jobs or user demands.

Barham et al. [1] propose Magpie, a toolchain for automatically extracting a system's workload under realistic operating conditions. Magpie is based on low-overhead instrumentation, incorporated to monitor the system and record fine-grained events generated by kernel, middleware and application components. In a more machine learning based approach, Cohen et al. [5] present a method for automatically extracting from running systems an indexable signature that distills the essential characteristics of the system state. Finally, in the same line Pan et al. [20] propose a tool for black-box diagnosis of MapReduce systems, aimed at discovering problems and bottlenecks.

Compared to all this related work, the GloBeM approach [16, 17] simplifies the analysis using a generic, global system model, therefore making both analysis and further decision-making easier. It also emphasizes less invasive monitoring and focuses on advanced knowledge discovery techniques that can directly be applied to improve the system.

The goal of this paper is to show how the GloBeM approach can be used to extract global state information and optimize the behavior of large-scale data-storage services under data-intensive workloads exhibiting a high degree of concurrency. To the best of our knowledge, such an approach has not been investigated so far in the area of distributed storage. To experiment this idea, a possible approach could be to use as an experimental playground well-established distributed file systems systems such as parallel and distributed file systems like GPFS [26], PVFS [21] or specialized file systems like GoogleFS [9] or HDFS [12], the storage system of the Hadoop [11] framework for MapReduce [6] programming. We have however preferred to apply the GloBeM approach to the BlobSeerr [18, 19] distributed data storage service. Compared to the systems mentioned above, BlobSeer introduces several novel techniques, like metadata decentralization and versioning based concurrency-control, with the goal of pushing the limits of exploiting parallelism at data-level even further. As supporting these features pushes further the complexity of the system as well, this makes

BlobSeer an ideal candidate to demonstrate our approach on: the goal of the GloBeM approach is precisely to provide a simple, easy-to-use model for complex distributed systems.

3 GloBeM: Global Behavior Modeling

GloBeM [16, 17] is a methodology for modeling the global behavior of a large-scale distributed system. Its main objective is to facilitate the understanding of the system behavior despite its complexity, by building an abstract, descriptive model of the global system state. This model presents the four following characteristics:

- **Specific state definition:** State characteristics and transition conditions should be unambiguously specified. The number of states should also be finite for usability reasons.
- **Stability:** The resulting model must be considerably consistent with the behavior of the environment (i.e. of the execution infrastructure, of the system usage patterns, etc.) over time. As distributed environments are naturally changing, it seems unrealistic to hope for stationarity and to try to find a definitive model. However, for a model to be useful, it must possess at least a certain degree of stability. A model that needs to be regenerate every time an event occurs in the system is simply unusable.
- **Simplicity:** The resulting model should be easy to understand and should provide basic and meaningful information about the system behavior. A complex model might be very precise, but too difficult to use.
- **Relevance to service:** The model states should be semantically related to the service provided by the system. This ensures that the observed behavior can be explained in terms of how this service is being provided. This makes it possible to determine if the conditions are acceptable and if the service provided meets expectations.

The GloBeM methodology specifies a set of necessary procedures in order to build such a model. It is strongly based on machine learning and other knowledge discovery techniques and divided in the three following phases (also shown in Fig. 1):

1. **Observing the system**: The system is observed using large-scale distributed systems monitoring techniques. At this point, every individual resource (e.g. storage node, metadata node, etc.) is monitored and the information is gathered. Then, the information obtained is represented in a more global way using a powerful information representation technique based on the construction of unsupervised Virtual Reality (VR) spaces [30, 31]. This technique enables one to create *easy-to-handle* representations of very complex information systems, by highlighting global aspects and thereby greatly simplifying the analysis.
2. **Analyzing data:** Once the monitoring information is properly represented, data mining techniques are applied in order to extract useful

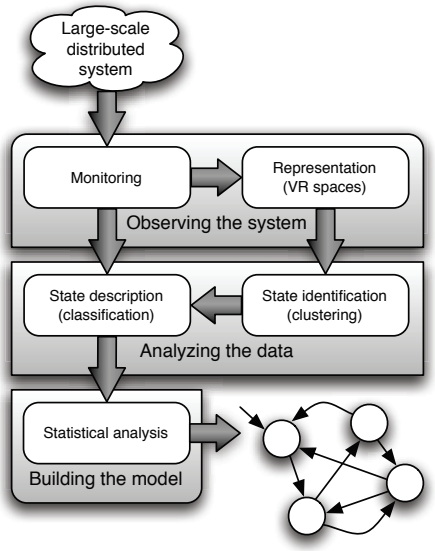


Figure 1: GloBeM phases overview

knowledge on behavior patterns and state related information. The result is a set of states and transitions that describe the system behavior from a single global perspective.

3. **Building the model:** Finally, a finite state machine model is constructed, which provides meaningful states and behavior information.

To sum up, applying this methodology results in a descriptive behavior model that provides a different outlook on system management: rather than considering separately the multiple entities that compose a complex distributed system, this approach focuses on global behavioral aspects and provides a *single entity* perspective. Such a behavior model introduces an innovative approach to the management of large-scale distributed environments. In this paper we illustrate its usefulness in the context of *distributed data management* on large-scale infrastructures. This context provides an excellent use case, precisely because of the complexity of large-scale distributed storage systems. Building a global, easy-to-use model in such a context significantly helps introducing simple, *global* management policies.

4 BlobSeer

As a case study, we chose to use the BlobSeer distributed data service as an experimental playground for investigating the expected benefits of the GloBeM approach. BlobSeer has specifically been designed to deal with the requirements of large-scale data-intensive distributed applications that process raw unstructured data.

Design overview BlobSeer consists of a series of distributed communicating processes, as shown in Fig. 2. In BlobSeer, data is stored as unstructured sequences of bytes called *BLOBs* (*Binary Large OBject*). Each BLOB is made up of fixed-size *chunks* that are distributed among *data providers*. *Clients* read, write and append data to/from BLOBs. Data-intensive distributed applications typically employ a large number of processing elements that access the BLOB concurrently. *Metadata* facilitates access to a range *(offset, size)* for any existing version of a BLOB, by associating such a range with the *data providers* where the corresponding chunks are located. Metadata is stored and managed on *metadata providers* through a decentralized, DHT-based infrastructure (see details below). A central *version manager*, is in charge of assigning versions and of exposing newly created versions to the readers in such way as to ensure consistency. Finally, a *provider manager*, decides which chunks are stored on which data providers when writes or appends are issued by the clients.

A client of BlobSeer manipulates a BLOB through a simple access interface that enables creating a BLOB, reading/writing a subsequence of *size* bytes in the BLOB starting at *offset* and appending a sequence of *size* bytes to the BLOB. The access interface is versioning-oriented: each time a write or append is performed by the client, a new snapshot of the BLOB is generated rather then overwriting any existing data. This snapshot is labeled with an incremental version and the client is allowed to read from any past snapshot of the BLOB by specifying its version.

Although each write or append generates a new BLOB version, only the differential patch is actually stored, so that storage space is saved at far as possible. The new snapshot physically shares all unmodified data and most of the associated metadata with the previous versions.

The goal of BlobSeer is to sustain *high throughput* under *heavy access concurrency* in reading, writing and appending. Three key design factors enable BlobSeer to address these requirements: *data striping*, *distributed metadata management* and *versioning-based concurrency control*.

Data striping Each BLOB is split into chunks of a fixed size which is specified at the time the BLOB is created. Typically the size of the chunks matches or is a multiple of the size of the data the client is expected to process in one step. These chunks are distributed all over the storage space providers of BlobSeer. A round-robin chunk allocation strategy is employed when writes and appends are issued, in order to evenly distribute the blocks among data providers, for load-balancing purposes. This has a major role in sustaining a high throughput when concurrent clients access different parts of the BLOB.

Metadata decentralization Since each BLOB is spread across a large number of storage space providers, BlobSeer needs to maintain metadata that maps subsequences of the BLOB to the corresponding chunks. In traditional approaches, this is the job of a centralized metadata server that is responsible for metadata maintenance and consistency [9, 26, 21, 12]. In contrast, BlobSeer uses a distributed metadata management scheme for two main reasons: firstly it avoids the bottleneck of accessing the same centralized node for metadata queries under heavy access concurrency and secondly it avoids having the metadata server act as a single point of failure.

Metadata is organized as a *distributed segment tree* [32]. This is essentially a binary tree in which each node is associated to a subsequence of the BLOB delimited by an offset and a size. Informally we say the node covers the range delimited by offset and size. Each leaf covers a single chunk. The left child of a non-leaf node covers the first half of the range covered by its parent and the right child covers the second half. The root covers the whole BLOB. Each tree node labeled with a BLOB version as well.

Thus, reading a subsequence from a BLOB means descending from the root that corresponds to the requested version towards the leaves that cover the subsequence and then fetching the corresponding chunks from the storage space providers. Writing or appending to the BLOB means first sending the new chunks to the storage space providers, then building a corresponding set of leaves labeled with the new version, and finally building the corresponding subtree labeled with the new version up to the root while linking to the tree nodes of the previous versions to offer the illusion of an independent snapshot.

To favor efficient concurrent access to metadata, tree nodes are distributed: they are stored on the metadata providers using a DHT (Distributed Hash Table). Each tree node is identified in the DHT by its version and by the range specified through the offset and the size it covers. Note that metadata decentralization has a significant impact on the global throughput, as demonstrated in [19]: it avoids the bottleneck created by concurrent accesses when a centralized metadata server is used (which is the case in most distributed file systems). A detailed description of the algorithms used to manage metadata can be found in [18].

Versioning-based concurrency control Versioning is not only provided at client level through the access interface, but is also leveraged by the concurrency control to increase the number of operations performed in parallel by avoiding synchronization as much as possible. The key idea is simple: thanks to versioning *any existing data or metadata is not overwritten*. This enables the readers to be completely decoupled from the writers: concurrent readers and writers will never interfere with each other because writers never modify an existing BLOB snapshot, not at data nor at metadata level (read/write concurrency). Thus from the reader point of view the BLOB snapshot is at all times in a consistent state and no synchronization is necessary, despite the concurrent writers.

Because data is never overwritten and only the difference with respect to the previous version is stored, writers/appenders also can send their chunks to the storage space providers independently of each other (write/write concurrency). It is at metadata level where the synchronization takes place and the chunks are consolidated into a new version. The detailed algorithms are available in [18].

Fault tolerance issues BlobSeer enables distributed data-intensive applications to scale to a large number of nodes, but many nodes means expecting faults to happen as part of normal functioning rather than as an exception. We call *fault* an abnormal state of a component of BlobSeer's architecture. We consider faults that are due to *failures* of the physical resources. In order to handle faults gracefully, two main requirements need to be satisfied. Firstly, *faults must be transparent* to the user: the application should continue running

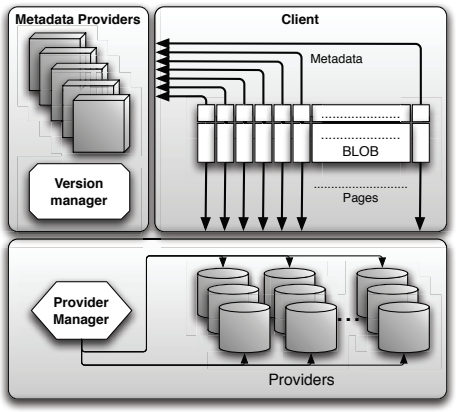


Figure 2: Architecture of BlobSeer

correctly as if no fault happened. Secondly, *faults must not impact performance significantly.*

To maintain data availability in spite of failures, BlobSeer replicates each chunk on multiple distinct providers. The provider manager is responsible for selecting the set of providers where to store the replicas for each newly chunk. The replication factor is configurable per BLOB at creation time. Such a simple scheme is acceptable because we leverage versioning. Since writes do not modify data in place, but rather generate new chunks that are included in new snapshots of the BLOB, implementing a simple replication that avoids complex consensus-based consistency protocols can be afforded as used in traditional systems. This is possible because readers are not aware of the effects of concurrent writes until a new snapshot is exposed for reading and therefore any potential inconsistent transient state is avoided by delaying the exposure of a new version until all the replicas have been written.

5 Improving QoS in BlobSeer using GloBeM

The high complexity of storage services in large-scale data-intensive settings makes it difficult to analyze the behavior of the system by explicitly considering every individual resource separately. The use of the global behavior modeling techniques described in Section 3 can substantially help by providing a valuable insight into the system's performance and evolution. GloBeM can automatically construct a descriptive model of BlobSeer's global behavior that can be used to identify bottlenecks and improve the overall performance and quality of service.

Experimental methodology To achieve the goal above we define the following methodology. We simulate a synthetic data-intensive workload on a BlobSeer instance running in a large-scale distributed setting for a long enough time so that a behavior can emerge. For realism, this running instance is subject to a failure-scenario generated by a failure-injection framework we implemented, which is responsible for simulating data providers going down and becoming available again according to typical failure patterns observed in such large-scale

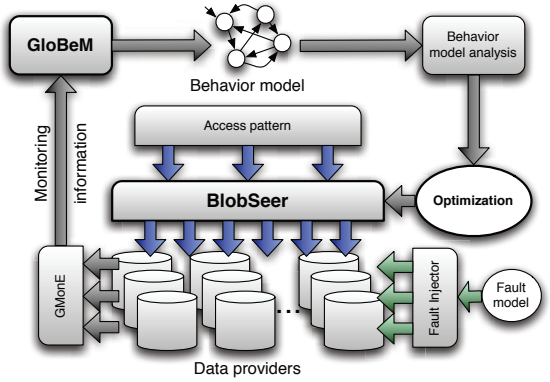


Figure 3: Improving QoS in BlobSeer by means of GloBeM

setups. We monitor BlobSeer throughout its run time and feed the monitoring information to GloBeM in order to automatically characterize BlobSeer's behavior. This in turn allows us to identify potential adjustments to BlobSeer that can be used to improve its performance and increase its quality of service. Fig. 3 shows a graphical description of the whole process.

Generating a data-intensive workload A typical scenario in large-scale data-intensive applications consists in continuously acquiring (and possibly updating) huge datasets of unstructured data while performing large-scale computations over the data. For example, crawling the web for new content such as text, audio, video may proceed in parallel with a processing phase for these information to build search index structures and aggregated statistics or discover new knowledge useful for Internet services or scientific applications [3]. For this reason, generating a typical data-intensive workload involves two aspects: i) a write access pattern that corresponds to constant data gathering and maintenance of data in the system and (in parallel) ii) a read access pattern that corresponds to the data processing.

As explained in [9], because managing a huge set of small files is not feasible, data is typically gathered in few but huge files. Moreover, experience with data-intensive applications has shown that these huge files are generated mostly by appending records concurrently and seldom overwriting any record. To reproduce this behavior in BlobSeer, we create a small number of BLOBs and have a set of clients generating and writing random data concurrently to the BLOBs. Each client predominantly appends and occasionally overwrites chunks of 64 MB to a randomly selected BLOB at random time intervals, sleeping meanwhile. The frequency of writes corresponds to an overall constant pressure of 1MB/s on each of the data provides of BlobSeer throughout the duration of the experiment.

Data intensive processing through MapReduce In order to model the data processing aspect, we consider workloads generated by MapReduce [6] applications, which typically scan the whole dataset in parallel and aggregate interesting information about it. This translates into a highly concurrent read

access pattern to the same BLOB. Note that writing the final end result is negligible, because most of the time it is a single aggregated value, such as the number of times a certain pattern occurred in the dataset. For this reason, we omit modeling writes of end results. To reproduce the highly concurrent read access pattern, we implemented clients that perform parallel reads of chunks of 64MB from the same BLOB version and then simulate a computation on this data by keeping the CPU busy. We keep an average I/O time to computation time ratio of 1:7 (which is typical for MapReduce applications). As it is highly likely that computations of chunks from the same BLOB take similar amount of time to complete, we need to compensate for this effect. To this end we adjusted reads to happen, unlike the case of writes, in bursts that put a more variable pressure on the data providers.

We execute both the data gathering and data processing concurrently in order to simulate a realistic setting where data is constantly analyzed while updates are processed in the background. We implemented the clients in such way as to target an overall write to read ratio of 1:10.

Simulating the failures Since real large-scale distributed environments are subject to failures, we implemented a data provider failure-injection framework that models failure patterns observed in real large-scale systems build from commodity hardware that run for long periods of time. We use the multi-state resource availability characterization study described in [22] in order to generate random failure scenarios for our experiments.

In our failure scenarios, each data provider is assigned a predefined behavior in time: when it is unavailable and when it is available. A transition from available to unavailable means killing the data provider, while a transition from unavailable to available means restarting the data provider. The individual behavior of all providers is generated in such way that the global behavior fits the results obtained in the study mentioned above.

Monitoring the data providers We periodically collect a wide range of parameters that describe the state of each data provider of the BlobSeer instance. For this task we use GMonE [25], a monitoring framework for large-scale distributed environments based on the *publish-subscribe* paradigm.

GMonE runs a process called *resource monitor* on every node to be monitored. Each such node publishes monitoring information to one or more *monitoring archives* at regular time intervals. These monitoring archives act as the subscribers and gather the monitoring information in a database, constructing a historical record of the system's evolution.

The *resource monitors* can be customized with *monitoring plugins*, which can be used to adapt the monitoring process to a specific scenario by selecting relevant monitoring information. We developed a plug-in for BlobSeer that is responsible for monitoring each provider and for pushing the following parameters into GMonE: number of read operations, number of write operations, free space available, CPU load and memory usage. These parameters represent the state of the provider at any specific moment in time. Every node running a data provider that is alive (was not rendered unavailable by the failure-injector) publishes this information each 45 seconds to a single central *monitoring archive* that stores the monitoring information for the whole experiment.

Building the global history record Once the monitoring information is gathered, an aggregation process is undertaken, in order to calculate global values. Mean and standard deviation values are calculated for each of the five previous metrics, producing general descriptors of the system global behavior. Additionally, the number of data providers available is included as an additional parameter. The resulting set of monitoring metrics (mean and standard deviation for each data provider metric plus the number of available data providers) provides a global historical record of the BlobSeer behavior.

Applying GloBeM to the collected data The historical data mentioned above is then fed into GloBeM in order to classify the global behavior of BlobSeer as a system into a set of states. Thanks to GloBeM, this process is fully automated and we obtain a characterization of the states in terms of the most important parameters that lead to this state. We collect client-side information as well: what operations were performed, if they were successful and how well the operations did perform in terms of performance measurements such as observed bandwidth from the client point of view.

Interpreting the state representation Armed with the client-side information and the characterizations of the states of BlobSeer, we can reason about aspects of BlobSeer that can be improved to cope with such data-intensive workloads. More precisely, we classify states into desirable states that offer good performance to the clients and undesirable states that offer poorer performance to the clients.

Improving BlobSeer Finally, the challenge is to improve the behavior of BlobSeer in such way as to avoid undesirable states. As this is completely dependent on the characterization of the states obtained by using GloBeM and part of our results, we detail this aspect in Section 6.2.3. For now, we stress that modeling the behavior of BlobSeer hard, realistic conditions applied simultaneously to recreate a real-life behavior is an inherently difficult problem. The use of GloBeM helped us to address this problem using automated global state identification. This approach enabled us to spot improvements that would have been very difficult to detect by empirical means.

6 Experimental evaluation

Section 5 describes how to apply GloBeM behavior modeling techniques to improve BlobSeer, optimizing it for a data-intensive access pattern. In the following section we describe the experimental procedures and results obtained to validate our proposal.

6.1 Experimental setup

We performed our experiments on the Grid'5000 [14] testbed, a highly configurable and controllable experimental Grid platform gathering 9 sites in France. We used the nodes of the Lille (130 nodes) and Orsay (275 nodes) clusters. The nodes are outfitted with x86_64 CPUs, and 2 GB of RAM. We measured raw buffered reads from the hard drives at 61.8MB/s on Lille and 53.2 MB/s on

Orsay, using the *hdparm* utility. Internode bandwidth is 1 Gbit/s (we measured 117.5 MB/s for TCP end-to-end sockets with MTU of 1500 B) and latency is 0.1 ms.

MapReduce-style computing systems are traditionally running on commodity hardware, collocating computation and storage on the same physical box. Recent proposals [27] advocate the use of converged networks to decouple the computation from storage in order to enable a more flexible and efficient datacenter design.

We aim at evaluating the benefits of applying global behavior modeling to BlobSeer in both scenarios. For this purpose, we use the Lille cluster to model collocation of computation and storage on the same physical node and the Orsay cluster to model decoupled computation and storage.

More specifically, in the case of Lille cluster we execute a version manager, a provider manager and 15 metadata providers on dedicated nodes. We use 110 further nodes from Lille, on each of which we codeploy a data provider with a client of BlobSeer that generates the data-intensive workload.

In the case of Orsay, we set up a version manager, a provider manager and 15 metadata providers on dedicated nodes. This time we deploy data providers on 120 dedicated nodes and use another 120 dedicated nodes to run the clients that generate the workload.

In both scenarios we deploy on each node that runs a data provider a GMonE resource monitor that is responsible to collect the monitoring data throughout the experimentation. Further, in each of the clusters we reserve a special node to act as the GMonE monitoring archive that collects the monitoring information from all resource monitors.

We will refer from now on to the scenario that models collocation of computation and storage on the Lille cluster simply as *setting A* and to the scenario that models decoupled computation and storage on the Orsay cluster as *setting B*.

6.2 Results

To evaluate the benefits of applying global behavior modeling to BlobSeer, we perform a multi-stage experimentation that involves: running an original BlobSeer instance under the data-intensive access pattern and failure scenario described in Section 5, interpreting the results by applying GloBeM, identifying an improvement and finally running an improved BlobSeer instance in the same conditions as the original instance and finally comparing the results and proving the improvement hinted by GloBeM was indeed successful in raising the performance of BlobSeer. This multi-stage experimentation is performed for both settings A and B described in Section 6.1.

6.2.1 Running the original BlobSeer instance

We ran a BlobSeer instance in both settings A and B, while the entire process was monitored using GMonE. Once the historical monitoring records were parsed successfully, we applied GloBeM to generate the global behavior model for each of the two experiments.

Both experiments ran for a fixed duration of 10 hours. The data-intensive workload accessed a total of $\simeq 11TB$ of data on setting A, out of which $\simeq 1.3TB$

Table 1: Global states - Setting A

parameter	State 1	State 2	State 3	State 4
Avg. read ops.	68.9	121.2	60.0	98.7
Read ops stdev.	10.5	15.8	9.9	16.7
Avg. write ops.	43.2	38.4	45.3	38.5
Write ops stdev.	4.9	4.7	5.2	7.4
Free space stdev.	3.1e7	82.1e7	84.6e7	89.4e7
Nr. of providers	107.0	102.7	96.4	97.2

Table 2: Global states - Setting B

parameter	State 1	State 2	State 3
Avg. read ops.	98.6	202.3	125.5
Read ops stdev.	17.7	27.6	21.9
Avg. write ops.	35.2	27.5	33.1
Write ops stdev.	4.5	3.9	4.5
Free space stdev.	17.2e6	13.0e6	15.5e6
Nr. of providers	129.2	126.2	122.0

was written and the rest read. Similarly, a total of $\simeq 17TB$ of data was generated on setting B, out of which $\simeq 1.5TB$ was written and the rest read.

6.2.2 Interpreting the monitoring results

GloBeM analysis produces a set of global states, indicating statistical information that characterizes each one of them. As explained in Section 3, the GloBeM process involves the use of a combination of information representation and machine learning techniques. Its objective is to identify similarities in the historical monitoring information and use them to infer behavior patterns as system states. A statistical analysis is then performed in order to obtain representative descriptors for each distinctive behavior that was detected. Tables 1 and 2 show the average values for the most representative monitoring metrics in each global state, for both settings A and B. As can be seen, GloBeM identified four possible states in the case of setting A and three in the case of setting B.

These behavior models were combined with additional metrics extracted from the client logs (as explained in Section 5), in order to identify a relationship between internal global behavior patterns (observed by GloBeM) and service quality and performance. We considered the number of read faults and effective read bandwidth to be relevant metrics from the client point of view.

Table 3: Average read bandwith - Setting A

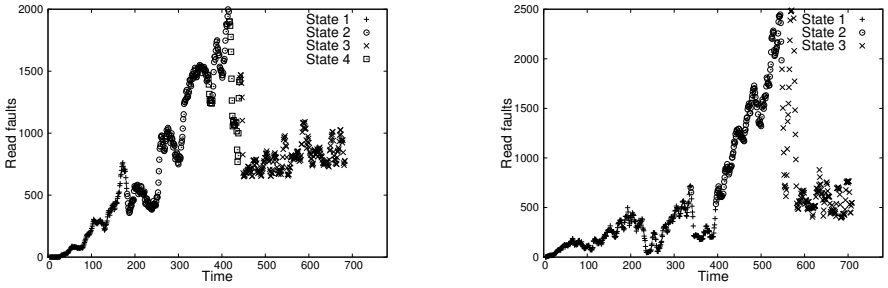
State 1	State 2	State 3	State 4
24.2	20.1	31.5	23.9

units are MB/s

Table 4: Average read bandwith - Setting B

State 1	State 2	State 3
50.7	35.0	47.0

units are MB/s



(a) Setting A (b) Setting B

Figure 4: Read faults: states are represented with different point styles

Average read bandwidths for each of the states are represented in Tables 3 and 4 for both settings A and B. Figures 4(a) and 4(b) depict evolution in time of the total number of read faults as observed by the clients for both scenarios. At this point it is important to remember that these are client related data and, therefore, neither read bandwidth nor failure information was available to GloBeM when identifying the states. Nevertheless, the different global patterns identified correspond to clearly different behavior in terms of client metrics, as Tables 3 and 4 and Figures 4(a) and 4(b) show.

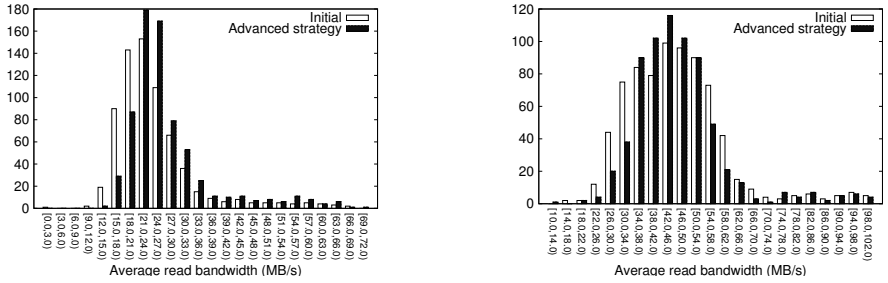
As previously described, the GloBeM analysis generated two global behavior models, each one corresponding to the behavior of BlobSeer in settings A and B. We performed further analysis using the effective read bandwidth and number of read faults as observed from the client point of view in order to classify the states of the behavior models into *desired* states (where the performance metrics are satisfactory) and undesired states (where the performance metrics can be improved).

In the case of setting A, *State 2* presents the lowest average read bandwidth ($\simeq 20MB/s$). It is also the state where most read faults occur, and where the failure pattern is more erratic. A similar situation occurs with setting B. In this case again *State 2* is the one with the lowest average bandwidth ($\simeq 35MB/s$) and the most erratic read fault behavior. We conclude these states (*State 2* in both settings A and B) to be *undesired*, because the worst quality of service is observed from the client point of view.

Considering now the global state characterization provided by GloBeM for both scenarios (Tables 1 and 2), a distinctive pattern can be identified for these *undesired* states: both have clearly the highest average number of read operations and, in the case of setting B specifically, a high standard deviation for the number of read operations. This indicates a state where the data providers are under heavy read load (hence the high average value) and the read operation completion times are fluctuating (hence the high standard deviation).

6.2.3 Improving BlobSeer

Now that we know where the problem is, we aim at improving BlobSeer by implementing a mechanism that tries to avoid reaching the *undesired* states described above. Since the system is under constant write load in all states for both settings A and B (Tables 1 and 2) we aim at reducing the total I/O



(a) Setting A (b) Setting B

Figure 5: Read bandwidth stability: distribution comparison

pressure on every data provider by avoiding to allocate providers under heavy read load to store new chunks generated by writers.

This in turn improves the read throughput but at the cost of a slightly less balanced chunk distribution. This eventually affects the throughput of future read operations on the newly written data. For this reason, avoiding writes on providers with heavy read loads is just an emergency measure to prevent reaching an *undesired* state. During normal functioning with non-critically high read loads, the original load-balancing strategy for writes can be used.

The average read operation characterization provided by GloBeM for *State 2*, which is the *undesired* state (both in settings A and B), is the key threshold to decide when a provider is considered to be under heavy read load and should not store new chunks. We implemented this idea in the chunk allocation strategy of the provider manager. Since data providers report periodically to the provider manager with statistics, we simply avoid choosing providers for which the average number of read operations goes higher than the threshold. We enable choosing those providers again when the number of read operations goes below this threshold.

6.2.4 Running the improved BlobSeer instance

The same experiments were again conducted in the exact same conditions, (for both settings A and B), using in this case the improved BlobSeer chunk allocation strategy. As explained, the purpose of this new strategy is to improve the overall quality of service by avoiding the undesirable states identified by GloBeM (*State 2* in both settings A and setting B).

As final measure of the quality of service improvement, a deeper statistical comparison of the average read bandwidth observed by the clients was done. Figures 5(a) and 5(b) show the read bandwith distribution for each experimental scenario. In each case, the values of the original and improved BlobSeer version are compared. Additionally, Table 5 shows the average and standard deviation observed in each experiment scenario.

The results seem to indicate a clear improvement (especially in setting A). However, in order to eliminate the possibility of reaching this conclusion simply because of different biases in the monitoring samples, we need further statistical assessment. In order to consider differences in values on Figures 5(a) and 5(b) and Table 5 as statistically meaningful, we need to ensure that the different

Table 5: Statistical descriptors for read bandwith (MB/s)

Scenario	mean (MB/s)	standard deviation
Setting A - Initial	24.9	9.6
Setting A - Advanced strategy	27.5	7.3
Setting B - Initial	44.7	10.5
Setting B - Advanced strategy	44.7	8.4

monitoring samples are in fact obtained from different probability distributions. This would certify that the quality of service improvement observed is real, and not a matter of simple bias.

We compared the read bandwidth observations using the Kolmogorov-Smirnov statistical test [29]. This technique works under the null hypothesis that the samples are drawn from the same distribution. A statistically meaningful result indicating that the null hypothesis is false[2] would validate the differences observed with the new BlobSeer strategy. The test results can be seen in Table 6. The values obtained clearly indicate that the null hypothesis is false in both cases, which validates the statistical relevance of the differences observed.

Table 6: Kolmogorov-Smirnov test results

Scenario	p-value
Setting A	$2.098e{-}14$
Setting B	0.004529

Finally, the results show a clear quality of service improvement in both settings A and B. In setting A, the average read bandwidth shows a 10% increase and, which is more important, the standard deviation was reduced substantially. This indicates a lesser degree of dispersion in the effective read bandwidth observed, and therefore a much more stable bandwidth (for which the difference between the expected bandwidth (the mean value) and the real bandwidth as measured by the client is lower). As it has been said, these read bandwidth mean and standard deviation improvements indicate an significant increase in the overall data access quality of service.

In setting B, the average read bandwidth remained stable, which is understandable given that, as explained in Section 6.1, we are close to the maximum physical hard drive transfer rate limit of the testbed characteristics and, therefore, achieving a higher value is very difficult. Nevertheless, the read bandwidth standard deviation was again significantly reduced, resulting in a much more stable data access and, therefore, improved data access quality of service.

7 Conclusions

To adequately support the requirements of distributed data-intensive applications on large-scale infrastructures, maintaining a high quality of service for the data storage system is crucial. The massively parallel data accesses issued by the application places a heavy burden on the storage service which has react

[2] In statistical hypothesis testing, the p-value is the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. For our work we have considered that a p-value ≤ 0.01 result in the test is statistically meaningful proof that the null hypothesis is false.

efficiently. To improve the quality of service provided by the storage service, we need to reason about its behavior in order to identify potential bottlenecks. However, the complexity of the system's behavior makes this problem difficult. A lot of different factors affect the behavior simultaneously: highly-concurrent data access patterns, long periods of service uptime, failures of physical components, the highly distributed nature of the storage service itself, etc.

This paper proposes a new approach to the management of QoS in a distributed storage system, based on global behavior modeling techniques to create a system abstraction and extract only relevant information. This approach automatically creates an easy-to-use, state-based system abstraction by extracting only relevant information from a large amount of data collected by monitoring the system. This representation highlights global behavior patterns, which provide an insight into the system's evolution and performance. Combined with client-side quality of service information, these behavior patterns can then be interpreted according to the level of quality of service provided. Undesirable states of the system can thus be identified, as well as potential improvements that can be applied in order to avoid undesired behavior patterns that exhibit poor quality of service.

We have successfully applied our approach to BlobSeer, a representative distributed storage service which aims at providing a high data throughput under heavy access concurrency. We evaluate this improvement through extensive evaluations on the Grid'5000 testbed using hard experimental conditions: highly-concurrent data access patterns, for long periods of service uptime, while supporting failures of the physical storage components. We used two settings, where computation jobs are executed on storage nodes and on separate nodes, respectively. In both settings we show substantial improvement in read bandwidth stability, thereby raising the overall quality of service provided by BlobSeer. Moreover, in the first setting we obtain an overall read bandwidth improvement of $\simeq 10\%$. Reaching the physical hard drive transfer rate is the only reason that prevents us from obtaining significant overall read bandwidth gains in the second setting as well.

As the benefits of improving the overall read bandwidth are obvious, we will not insist on the significance of this aspect. In contrast, we stress the benefits of improving the bandwidth stability: a higher quality of service at this level makes the cost of access operations more predictable, improves the efficiency of scheduling algorithms used by data-intensive processing frameworks and helps optimizing the overall application throughput. We plan to continue our work in this direction, which we expect to reveal additional substantial gains.

References

[1] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, pages 259–272, 2004.

[2] Carmen Bratosin, Wil M. P. van der Aalst, Natalia Sidorova, and Nikola Trcka. A reference model for grid architectures and its analysis. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 5331 of *Lecture Notes in Computer Science*, pages 898–913. Springer, 2008.

[3] Randal E. Bryant. Data-intensive supercomputing: The case for disc. Technical report, CMU, 2007.

[4] Kyu Cheol Cho, Tae Young Kim, and Jong Sik Lee. User demand prediction-based resource management model in grid computing environment. In *Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology (ICHIT '08)*, pages 627–632, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. *SIGOPS Oper. Syst. Rev.*, 39(5):105–118, 2005.

[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.

[8] Jack J. Dongarra and Wolfgang Gentzsch, editors. *Computer benchmarks*. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 1993.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS - Operating Systems Review*, 37(5):29–43, 2003.

[10] Yuri Gurevich. Evolving Algebras: An Attempt to Discover Semantics. In *EATCS Bulletin*, volume 43, pages 264–284. European Assoc. for Theor. Computer Science, February 1991.

[11] The Apache Hadoop Project. http://www.hadoop.org.

[12] HDFS. The Hadoop Distributed File System. http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html.

[13] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, 2007.

[14] Yvon Jégou, Stephane Lantéri, Julien Leduc, Melab Noredine, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Touche Iréa. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006.

[15] Hui Li, David Groep, and Lex Wolters. Mining performance data for metascheduling decision support in the grid. *Future Gener. Comput. Syst.*, 23(1):92–99, 2007.

[16] Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. The grid as a single entity: Towards a behavior model of the whole grid. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 5331 of *Lecture Notes in Computer Science*, pages 886–897. Springer, 2008.

[17] Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero. Finding order in chaos: a behavior model of the whole grid. *Concurrency and Computation: Practice and Experience*, page In press., 2009.

[18] Bogdan Nicolae, Gabriel Antoniu, and Luc Bougé. BlobSeer: How to enable efficient versioning for large object storage under heavy access concurrency. In *Proc. 2nd Workshop on Data Management in Peer-to-Peer Systems (DAMAP'2009)*, Saint Petersburg, Russia, March 2009. Held in conjunction with EDBT'2009.

[19] Bogdan Nicolae, Gabriel Antoniu, and Luc Bougé. Enabling high data throughput in desktop grids through decentralized data and metadata management: The blobseer approach. In *Proc. 15th International Euro-Par Conference on Parallel Processing (Euro-Par '09)*, volume 5704 of *Lect. Notes in Comp. Science*, pages 404–416, Delft, The Netherlands, 2009. Springer-Verlag.

[20] Xinghao Pan, Soila Kavulya, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Ganesha: Black-box diagnosis for mapreduce systems. In *Proceedings of the Second Workshop on Hot Topics in Measurement & Modeling of Computer Systems*, Seattle, WA, USA, 2009.

[21] PVFS. Parallel virtual file system, version 2. http://pvfs2.org/.

[22] Brent Rood and Michael J. Lewis. Multi-state grid resource availability characterization. In *GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, pages 42–49, Washington, DC, USA, 2007. IEEE Computer Society.

[23] Brent Rood and Michael J. Lewis. Resource availability prediction for improved grid scheduling. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience (e-Science 2008)*, pages 711–718, Washington, DC, USA, 2008. IEEE Computer Society.

[24] Brent Rood and Michael J. Lewis. Scheduling on the grid via multi-state resource availability prediction. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*, pages 126–135. IEEE, 2008.

[25] A. Sánchez. *Autonomic high performance storage for grid environments based on long term prediction. Chapter 9.2.* PhD thesis, Universidad Politécnica de Madrid, 2008.

[26] Frank B. Schmuck and Roger L. Haskin. GPFS: A shared-disk file system for large computing clusters. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 231–244. USENIX Association, 2002.

[27] Jeffrey Shafer, Scott Rixner, and Alan L. Cox. Datacenter storage architecture for mapreduce applications. In *ACLD: Workshop on Architectural Concerns in Large Datacenters*, 2009.

[28] Warren Smith, Ian T. Foster, and Valerie E. Taylor. Predicting application run times with historical information. *J. Parallel Distrib. Comput.*, 64(9):1007–1016, 2004.

[29] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.

[30] J. J. Valdés. Similarity-based heterogeneous neurons in the context of general observational models. *Neural Network World*, 12:499–508, 2002.

[31] J. J. Valdés. Virtual reality representation of information systems and decision rules. *Lecture Notes in Artificial Intelligence*, 2639:615–618, 2003.

[32] C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed Segment Tree: Support range query and cover query over DHT. In *Proceedings of the Fifth International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, California, 2006.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399